databases. This appears to be an issue of terminology, not substance. Applicant has amended the independent claims to clarify that the operands of a join are database tables. Additionally, applicant has amended dependent claims 19, 24, 29, and 32 accordingly to refer back to the term "database table" instead of "database." (Claim 32 has also been amended to correct a minor typographical oversight by adding the word "the" before "chemical similarity join".) Support for these amendments can be found at various pages of the application, including page 24 ("The information in these tables can be combined to produce information of interest using a database 'join'. A join associates rows in one table to those in another based on relationships between values of certain columns for the records"); see also pp. 31-32. This amendment does not affect the substance or scope of the claims, but obviates the new matter rejection, since performing a join on database tables is clearly within the scope of the disclosure.

The Meaning of the Term-of-Art "Join"

The Examiner's section 102 and 103 rejections of claims 1 and 10-33 over Maslyn, Schmit, and Cramer, appear to be based on a fundamental misunderstanding of the term "join," which is a feature of all of the claims. In particular, the Examiner states on pages 3 of the Office Action that "[d]atabase sequences may be partitioned into multiple tables. Thus, a search of a target sequence would result in *joining* of rows of multiple databases to provide the results to the user" (emphasis added). Furthermore, at various places in the Office Action, the Examiner refers to "partitioning" a database table into multiple tables in order to search selected parts of the database. (See pages 4-5). It appears that the Examiner is saying that if a user partitioned a single database table into two tables, searched each partition separately for select rows, and then combined the results, this combination of the results would be a "join" of two tables. This, however, is not what the claim term "join" means in the art of database systems.

As noted in the Application, "join ... is a term of art." (Page 6, lines 9-14.) In particular, in the field of database systems, the operation join (written as "⋈"), is a binary operation performed on two relational database tables. In particular, if A and B are

two database tables, then $A \bowtie_\Theta B$ (where Θ is a logical predicate), yields a new table whose structure is different from either A or B. Specifically, if C_A is the number of columns in A, and C_B is the number of columns in B, then $A \bowtie_\Theta B$ is a new table that has $C_A + C_B$ columns. Additionally, if R_A and R_B are the number of rows in A and B, respectively, the new table produced by the join has up to a maximum of $R_A \cdot R_B$ rows; in fact, $A \bowtie_\Theta B$ includes all of the rows that satisfy the predicate Θ and that can be created by combining a row of A with a row of B. (A "predicate" is a statement that must be true about a row in order for the row to be included in the join result.) It is important to note that the result of the join *is not merely the result of searching or "partitioning" the tables A and B, as suggested by the Examiner*. There is no way that searching or partitioning tables A and B can produce the same results as $A \bowtie_\Theta B$.

The following is an example of a join. Suppose that Table A is a list of employees and their departments:

Employee	**Department**
Abigail	Education
Jack	Production
Pam	Legal
Peter	Legal

and that Table B is a list of the building that each department is located in:

Department	**Building**
Legal	50
Education	50
Production	4

Then $A \bowtie_\theta B$, where θ is the predicate "A.Department=B.Department $\wedge$ Building=50", yields the following table:

Employee	A.Department	B.Department	Building
Abigail	Education	Education	50
Pam	Legal	Legal	50
Peter	Legal	Legal	50

In this example, the predicate Θ is "A.Department=B.Department ∧ Building=50", which means that for a row to be included in the join result, the following two conditions must be true: (1) the value in the department column from table A must match the value from the department column in table B, and (2) the value in the building column must be 50. All of the rows in the join result shown above satisfy this predicate. The row:

Abigail	Education	Legal	50

can also be formed by combining a row of A with a row of B, but this row is not included in the join result, because it does not satisfy the predicate (i.e., A.department ("education") does not match B.department ("legal")). This example demonstrates that, contrary to the Examiner's argument, the particular nature of the join operation is not the same as searching or partitioning.

It can be seen that $A \bowtie_\theta B$ is a new table, which is structurally different from either A or B, and provides new information that could not be obtained merely by searching A and B (or by "partitioning" A and B, as the Examiner suggests). In particular, the new table produced by the join provides a list of which employees are located in building 50 – a result that cannot be obtained just by searching (or "partitioning") tables A and B.

The meaning of the term "join" is well-known in the art; it is thoroughly described in Jeffrey D. Ullman & Jennifer Widom, *A First Course in Database Systems*, pp. 173-94 (1997) (copy attached).

The Cramer and Schmit references do not teach a join. While the Maslyn reference mentions the term "join," (e.g., at FIG. 2B), the Examiner has not cited the portions of Maslyn that discuss a join (and, as explained below, even Maslyn does not

discuss the particular type of similarity join that is recited in the claims). Instead, the Examiner has cited the Cramer, Schmit, and Maslyn references for the teachings of searching and partitioning. As explained above, searching and partitioning are not the same thing as a join, nor do they suggest or render obvious the join operation.

The Similarity Join Recited in the Claims Defines Over the Prior Art

As explained above and in applicants' response to the prior Office Action, the Cramer and Schmitt references have nothing to do with the binary join operation. Moreover, while Maslyn mentions a join, the Examiner has cited only Maslyn's for its alleged teaching of a search and partition, not a join; in any event, Maslyn does not teach the similarity join recited in the claims. Applicants submit that the cited references do not teach or suggest any of the claims, since each of the claims calls for a similarity join. Moreover, the similarity join feature is not obvious over these references. To the extent that the Examiner argues that a join is merely an obvious application of "partitioning" a database, the preceding section demonstrates that this reasoning is simply incorrect. Accordingly, all of the claims are patentable over the references cited by the Examiner.

Finally, it should be noted that even if Maslyn had been cited for its mention of a join, the particular type of join recited in the claims patentably defines over prior art join operations, such as those mentioned in Maslyn or in the Ullman & Widom reference. Claims 1 and 23 call for a "chemical similarity join," and claim 10 calls for a "fuzzy similarity join." The generic join described above combines rows of two tables based on whether the combination satisfies some arbitrary predicate θ. For example, θ may call for a particular columnar value (e.g., "building = 50"), equality among columnar values (e.g., "A.department = B.department"), or some other standard condition. The claims, on the other hand, call for a join operation that combines rows based on *similarity* among the items that the rows represent. For example, claim 1 calls for performing a chemical similarity join between first and second database tables based on whether the similarity between compounds in the first and second database tables are within a specified neighborhood range. Claim 10 calls for performing a fuzzy similarity join that correlates rows of the two tables based on similar properties. Claim 23 calls for

performing a chemical similarity join to identify in the first table a compound whose property is similar to that of a target compound contained in the second table. Joining tabular rows based on *similarity* is a feature that is not taught or suggested in the prior art.

Thus, all of the independent claims are patentable over the prior art. Additionally, all of the dependent claims are patentable at least by reason of their dependency.

CONCLUSION

For all the foregoing reasons, Applicants submit that the claims, as amended, are patentable over the prior art of record. Applicants thus request that the section 112, 102, and 103 rejections of the pending claims be withdrawn, and submit that this case is now in condition for allowance. An early Notice of Allowance is earnestly solicited.

Respectfully submitted,

Peter M. Ullman
Registration No. 43,963

Dated: 10-2-02

WOODCOCK WASHBURN, LLP
One Liberty Place – 46th Floor
Philadelphia, PA 19103
Tel: (215) 568-3100
Fax: (215) 568-3439

VERSION WITH MARKINGS TO SHOW CHANGES MADE

1. (Twice Amended) A computer-based method for facilitating the selection of chemical compounds, comprising:

receiving an identification of a target compound and a neighborhood range from a user;

performing a chemical similarity join to identify compounds of interest within said neighborhood range of said target compound, said chemical similarity join being performed on:

a first database table that stores information about one or more compounds including said target compound; and

a second database table that stores information about a plurality of compounds,

wherein said chemical similarity join combines a compound from said first database table with a compound from said second database table based on whether the level of similarity between the compound from the first database table and the compound from the second database table is within said neighborhood range; and

providing results of said chemical similarity join to the user.

10. (Thrice amended) A computer-based method for retrieving information that is based upon at least one similarity among entities in a plurality of [databases] database tables, the method comprising:

identifying a target item, wherein a first of the database tables includes a row that identifies the target item;

using a computer to perform a fuzzy similarity join on the first database table and a second database table to correlate rows of the first and second [databases] tables that have similar properties; and

retrieving at least one item from the result of the join, wherein the retrieved item comprises at least one item having a property similar to a property of the target item.

19. (Amended) A method according to claim 11 further comprising eliminating test items from [the database] at least one of the first and second database tables by selection of user-defined criteria for non-desired items.

23. (Twice Amended) A computer-based method for identifying, from a first database table comprising chemical compounds, at least one chemical compound having at least one property similar to a target chemical compound, the method comprising:

identifying a property of a target chemical compound; and

using a computer to perform a chemical similarity join on the first database table and a second database table that includes the target compound to identify at least one [database] chemical compound in the first database table that has a property similar to the property of the target chemical compound.

24. (Amended) A method according to claim 23 wherein a user is informed of the identification of at least one [database] chemical compound in the first database table that has a property similar to the property of the target chemical compound.

29. (Amended) A method according to claim 23 wherein the similarity between the properties of the target chemical compound and [the database] a chemical compound in the first database table is determined using at least one parameter from the group consisting of a Tanimoto coefficient and a Molecular hologram.

32. (Amended) A method according to claim 23 further comprising excluding at least one [database] chemical compound in the first database table from the chemical similarity join by selecting user-defined exclusion criteria for non-desirable compound features.

Chapter 4

Operations in the Relational Model

In this chapter we begin the study of databases from the point of view of the user. Often, the principal issue for the user is *querying* the database, that is, writing programs that answer questions about the current instance of the database. In this chapter, we shall study the question of database queries from an abstract point of view, defining the principal query operators.

While ODL uses methods that, in principle, can perform any operation on data, and the E/R model does not embrace a specific way of manipulating data. the relational model has a concrete set of "standard" operations on data. Thus, our study of database operations in the abstract will focus on the relational model and its operations. These operations can be expressed in either an algebra, called "relational algebra," or in a form of logic, called "Datalog." We shall learn each of these notations in this chapter.

Later chapters let us see the languages and features that today's commercial database systems offer the user. The abstract query operators will appear primarily as operations in the SQL query language discussed in Chapters 5 through 7. However, they also appear in the OQL language mentioned in Chapter 8.

4.1 An Algebra of Relational Operations

To begin our study of operations on relations, we shall learn about a special algebra, called *relational algebra*, that consists of some simple but powerful ways to construct new relations from old ones. *Expressions* in relational algebra start from relations as operands; relations can either be represented by their name (e.g., R or `Movie`) or represented explicitly as a list of their tuples. We can then build progressively more complex expressions by applying any of the operators to be described below, either to relations or to simpler expressions of relational

algebra. A *query* is an expression of relational algebra. Thus, relational algebra is our first concrete example of a query language.

The operations of relational algebra fall into four broad classes:

1. The usual set operations — union, intersection, and difference — applied to relations.
2. Operations that remove parts of a relation: "selection" eliminates some rows (tuples), and "projection" eliminates some columns.
3. Operations that combine the tuples of two relations, including "Cartesian product," which pairs the tuples of two relations in all possible ways, and various kinds of "join" operations, which selectively pair tuples from two relations.
4. An operation called "renaming" that does not affect the tuples of a relation, but changes the relation schema, i.e., the names of the attributes and/or the name of the relation itself.

These operations are not enough to do any possible computation about relations; in fact they are quite limited. However, they capture much of what we really want to do with databases, and they form a large part of the standard relational query language SQL, as we shall see in Chapter 5. We shall, however, discuss briefly in Sections 4.6 and 4.7 some of the computational capabilities that are present in real query languages like SQL and yet are not part of relational algebra.

4.1.1 Set Operations on Relations

The three most common operations on sets are union, intersection, and difference. We assume the reader is familiar with these operations, which are defined as follows on arbitrary sets R and S:

- $R \cup S$, the *union* of R and S, is the set of elements that are in R or S or both. An element appears only once in the union even if it is present in both R and S.
- $R \cap S$, the *intersection* of R and S, is the set of elements that are in both R and S.
- $R - S$, the *difference* of R and S, is the set of elements that are in R but not in S. Note that $R - S$ is different from $S - R$; the latter is the set of elements that are in S but not in R.

When we apply these operations to relations, we need to put some conditions on R and S:

1. R and S must have schemas with identical sets of attributes.

2. Before we compute the set-theoretic union, intersection, or difference of sets of tuples, the columns of R and S must be ordered so that the order of attributes is the same for both relations.

Sometimes we would like to take the union, intersection, or difference of relations that have the same number of attributes but use different names for their attributes. If so, we may use the renaming operator discussed in Section 4.1.8 to change the schema of one or both relations and give them the same set of attributes.

name	*address*	*gender*	*birthdate*
Carrie Fisher	123 Maple St., Hollywood	F	9/9/99
Mark Hamill	456 Oak Rd., Brentwood	M	8/8/88

Relation R

name	*address*	*gender*	*birthdate*
Carrie Fisher	123 Maple St., Hollywood	F	9/9/99
Harrison Ford	789 Palm Dr., Beverly Hills	M	7/7/77

Relation S

Figure 4.1: Two relations

Example 4.1: Suppose we have the two relations R and S, instances of the relation MovieStar of Section 3.9. Current instances of R and S are shown in Fig. 4.1. Then the union $R \cup S$ is

name	*address*	*gender*	*birthdate*
Carrie Fisher	123 Maple St., Hollywood	F	9/9/99
Mark Hamill	456 Oak Rd., Brentwood	M	8/8/88
Harrison Ford	789 Palm Dr., Beverly Hills	M	7/7/77

Note that the two tuples for Carrie Fisher from the two relations appear only once in the result.

The intersection $R \cap S$ is

name	*address*	*gender*	*birthdate*
Carrie Fisher	123 Maple St., Hollywood	F	9/9/99

Now, only the Carrie Fisher tuple appears, because only it is in both relations.

The difference $R - S$ is

name	*address*	*gender*	*birthdate*
Mark Hamill	456 Oak Rd., Brentwood	M	8/8/88

That is, the Fisher and Hamill tuples appear in R and thus are candidates for $R - S$. However, the Fisher tuple also appears in S and so is not in $R - S$. □

4.1.2 Projection

The *projection* operator is used to produce from a relation R a new relation that has only some of R's columns. The value of expression $\pi_{A_1,A_2,\ldots,A_n}(R)$ is a relation that has only the columns for attributes $A_1, A_2, \ldots, A_n$ of R. The schema for the resulting value is the set of attributes $\{A_1, A_2, \ldots, A_n\}$, which we conventionally show in the order listed.

title	*year*	*length*	*inColor*	*studioName*	*producerC#*
Star Wars	1977	124	true	Fox	12345
Mighty Ducks	1991	104	true	Disney	67890
Wayne's World	1992	95	true	Paramount	99999

Figure 4.2: The relation Movie

Example 4.2: Consider the relation Movie with the relation schema described in Section 3.9. An instance of this relation is shown in Fig. 4.2. We can project this relation onto the first three attributes with the expression

$$\pi_{title,year,length}(\texttt{Movie})$$

The resulting relation is

title	*year*	*length*
Star Wars	1977	124
Mighty Ducks	1991	104
Wayne's World	1992	95

As another example, we can project onto the attribute inColor with the expression $\pi_{inColor}(\texttt{Movie})$. The result is the single-column relation

inColor
true

Notice that there is only one tuple in the resulting relation, since all three tuples of Fig. 4.2 have the same value in their component for attribute inColor. □

4.1.3 Selection

The *selection* operator, applied to a relation R, produces a new relation with a subset of R's tuples. The tuples in the resulting relation are those that satisfy some condition C that involves the attributes of R. We denote this operation $\sigma_C(R)$. The schema for the resulting relation is the same as R's schema, and we conventionally show the attributes in the same order as we use for R.

C is a conditional expression of the type that we are familiar with from conventional programming languages; for example, conditional expressions follow the keyword `if` in programming languages such as C or Pascal. The only difference is that the operands in C are either constants or attributes of R. We apply C to each tuple t of R by substituting, for each attribute A appearing in condition C, the component of t for attribute A. If after substituting for each attribute of C the condition C is true, then t is one of the tuples that appear in the result of $\sigma_C(R)$; otherwise t is not in the result.

Example 4.3: Let the relation `Movie` be as in Fig. 4.2. Then the value of expression $\sigma_{length \geq 100}(\texttt{Movie})$ is

title	*year*	*length*	*inColor*	*studioName*	*producerC#*
Star Wars	1977	124	true	Fox	12345
Mighty Ducks	1991	104	true	Disney	67890

The first tuple satisfies the condition $length \geq 100$ because when we substitute for *length* the value 124 found in the component of the first tuple for attribute `length`, the condition becomes $124 \geq 100$. The latter condition is true, so we accept the first tuple. The same argument explains why the second tuple of Fig. 4.2 is in the result.

The third tuple has a `length` component 95. Thus, when we substitute for *length* we get the condition $95 \geq 100$, which is false. Hence the last tuple of Fig. 4.2 is not in the result. □

Example 4.4: Suppose we want the set of tuples in the relation

```
Movie(title, year, length, inColor, studioName, producerC#)
```

that represent Fox movies at least 100 minutes long. We can get these with a more complicated condition, involving the `AND` of two subconditions. The expression is

$$\sigma_{length \geq 100 \texttt{ AND } studioName = \texttt{'Fox'}}(\texttt{Movie})$$

The tuple

title	*year*	*length*	*inColor*	*studioName*	*producerC#*
Star Wars	1977	124	true	Fox	12345

is the only one in the resulting relation. □

A	B
1	2
3	4

Relation R

B	C	D
2	5	6
4	7	8
9	10	11

Relation S

A	$R.B$	$S.B$	C	D
1	2	2	5	6
1	2	4	7	8
1	2	9	10	11
3	4	2	5	6
3	4	4	7	8
3	4	9	10	11

Result $R \times S$

Figure 4.3: Two relations and their Cartesian product

4.1.4 Cartesian Product

The *Cartesian product* (or just *product*) of two sets R and S is the set of pairs that can be formed by choosing the first element of the pair to be any element of R and the second an element of S. This product is denoted $R \times S$. When R and S are relations, the product is essentially the same. However, since the members of R and S are tuples, usually consisting of more than one component, the result of pairing a tuple from R with a tuple from S is a longer tuple, with one component for each of the components of the constituent tuples. The components from R precede the components from S in this order.

The relation schema for the resulting relation is the union of the schemas for R and S. However, if R and S should happen to have some attributes in common, then we need to invent new names for at least one of each pair of identical attributes. To disambiguate an attribute A that is in the schemas of both R and S, we use $R.A$ for the attribute from R and $S.A$ for the attribute from S.

Example 4.5: For conciseness, let us use an abstract example that illustrates the product operation. Let relations R and S have the schemas and tuples shown in Fig. 4.3. Then the product $R \times S$ consists of the six tuples shown in that figure. Note how we have paired each of the two tuples of R with each of the three tuples of S. Since B is an attribute of both schemas, we have used $R.B$ and $S.B$ in the schema for $R \times S$. The other attributes are unambiguous, and their names appear in the resulting schema unchanged. □

4.1.5 Natural Joins

More often than we want to take the product of two relations, we find a need to *join* them by pairing only those tuples that match in some way. The simplest sort of match is the *natural join* of two relations R and S, denoted $R \bowtie S$, in which we pair only those tuples from R and S that agree in whatever attributes are common to the schemas of R and S. More precisely, let $A_1, A_2, \ldots, A_n$ be the attributes in both the schema of R and the schema of S. Then a tuple r from R and a tuple s from S are successfully paired if and only if r and s agree on each of the attributes $A_1, A_2, \ldots, A_n$.

If the tuples r and s are successfully paired in the join $R \bowtie S$, then the result of the pairing is a tuple, called the *joined tuple*, with one component for each of the attributes in the union of the schemas of R and S. The joined tuple agrees with tuple r in each attribute in the schema of R, and it agrees with s in each attribute in the schema of S. Since r and s are successfully paired, we know that r and s agree on attributes that are in the schemas of both R and S. Thus, it is possible for the joined tuple to agree with both r and s on those attributes that are in both schemas. The construction of the joined tuple is suggested by Fig. 4.4.

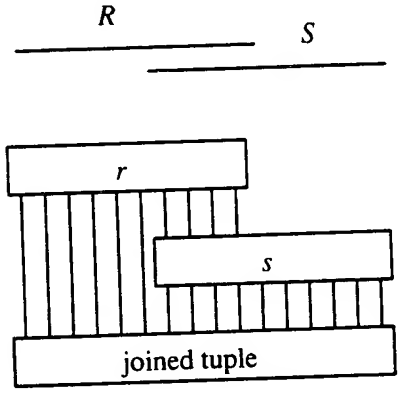

Figure 4.4: Joining tuples

Note also that this join operation is the same one that we used in Section 3.7.6 to recombine relations that had been projected onto two subsets of

their attributes. There the motivation was to explain why BCNF decomposition made sense. In Section 4.1.7 we shall see another use for the natural join: combining two relations so that we can write a query that relates attributes of each.

Example 4.6: The natural join of the relations R and S from Fig. 4.3 is

A	B	C	D
1	2	5	6
3	4	7	8

The only attribute common to R and S is B. Thus, to pair successfully, tuples need only to agree in their B components. If so, the resulting tuple has components for attributes A (from R), B (from either R or S), C (from S), and D (from S).

In this example, the first tuple of R successfully pairs with only the first tuple of S; they share the value 2 on their common attribute B. This pairing yields the first tuple of the result: $(1,2,5,6)$. The second tuple of R pairs successfully only with the second tuple of S, and the pairing yields $(3,4,7,8)$. Note that the third tuple of S does not pair with any tuple of R and thus has no effect on the result of $R \bowtie S$. A tuple that fails to pair with any tuple of the other relation in join is sometimes said to be a *dangling tuple.* □

Example 4.7: The previous example does not illustrate all the possibilities inherent in the natural join operator. For example, no tuple paired successfully with more than one tuple, and there was only one attribute in common to the two relation schemas. In Fig. 4.5 we see two other relations, U and V, that share two attributes between their schemas, B and C. We also show an instance in which one tuple joins with several tuples.

For tuples to pair successfully, they must agree in both the B and C components. Thus, the first tuple of U pairs successfully with the first two tuples of V, while the second and third tuples of U pair successfully with the third tuple of V. The result of these four pairings is shown in Fig. 4.5. □

4.1.6 Theta-Joins

The natural join forces us to pair tuples using one specific condition. While this way, equating shared attributes, is the most common basis on which relations are joined, it is sometimes desirable to pair tuples from two relations on some other basis. For that purpose, we have a related notation called the *theta-join*; the "theta" refers to an arbitrary condition, which we shall represent by C rather than θ.

The notation for a theta-join of relations R and S based on condition C is $R \bowtie_C S$. The result of this operation is constructed as follows:

1. Take the product of R and S.

A	B	C
1	2	3
6	7	8
9	7	8

Relation U

B	C	D
2	3	4
2	3	5
7	8	10

Relation V

A	B	C	D
1	2	3	4
1	2	3	5
6	7	8	10
9	7	8	10

Result $U \bowtie V$

Figure 4.5: Natural join of relations

2. Select from the product only those tuples that satisfy the condition C.

As with the product operation, the schema for the result is the union of the schemas of R and S, with "R." or "S." prefixed to attributes if necessary to indicate from which schema the attribute came.

Example 4.8: Consider the operation $U \bowtie_{A<D} V$, where U and V are the relations from Fig. 4.5. We must consider all nine pairs of tuples, one from each relation, and see whether the A component from the U-tuple is less than the D component of the V-tuple. The first tuple of U, with an A component of 1, successfully pairs with each of the tuples from V. However, the second and third tuples from U, with A components of 6 and 9, respectively, pair successfully with only the last tuple of V. Thus, the result has only five tuples, constructed from the five successful pairings. This relation is shown in Fig. 4.6. □

Notice that the schema for the result in Fig. 4.6 consists of all six attributes, with U and V prefixed to their respective occurrences of attributes B and C to distinguish them. Thus, the theta-join contrasts with natural join, since in the

A	$U.B$	$U.C$	$V.B$	$V.C$	D
1	2	3	2	3	4
1	2	3	2	3	5
1	2	3	7	8	10
6	7	8	7	8	10
9	7	8	7	8	10

Figure 4.6: Result of $U \bowtie_{A<D} V$

latter common attributes are merged into one copy. Of course it makes sense to do so in the case of the natural join, since tuples don't pair unless they agree in their common attributes. In the case of a theta-join, there is no guarantee that compared attributes will agree in the result, since the comparison operator might not be $=$.

Example 4.9: Here is a theta-join on the same relations U and V that has a more complex condition:

$$U \bowtie_{A<D \text{ AND } U.B \neq V.B} V$$

That is, we require for successful pairing not only that the A component of the U-tuple be less than the D component of the V-tuple, but that the two tuples disagree on their respective B components. The tuple

A	$U.B$	$U.C$	$V.B$	$V.C$	D
1	2	3	7	8	10

is the only one to satisfy both conditions, so this relation is the result of the theta-join above. □

4.1.7 Combining Operations to Form Queries

If all we could do was to write single operations on one or two relations as queries, then relational algebra would not be as useful as it is. However, relational algebra, like all algebras, allows us to form expressions of arbitrary complexity by applying operators either to given relations or to relations that are the result of applying one or more relational operators to relations.

One can construct expressions of relational algebra by applying operators to subexpressions, using parentheses when necessary to indicate grouping of operands. It is also possible to represent expressions as expression trees; the latter often are easier for us to read, although they are less convenient as a machine-readable notation.

Example 4.10: Let us reconsider the decomposed `Movie` relation of Example 3.32. Suppose we want to know "What are the titles and years of movies made by Fox that are at least 100 minutes long?" One way to compute the answer to this query is:

1. Select those `Movie` tuples that have $length \geq 100$.
2. Select those `Movie` tuples that have $studioName =$ `'Fox'`.
3. Compute the intersection of (1) and (2).
4. Project the relation from (3) onto attributes `title` and `year`.

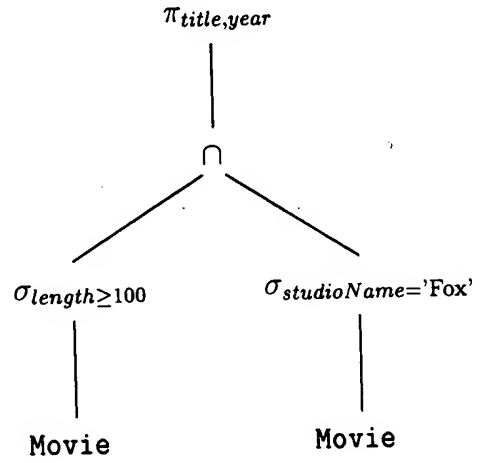

Figure 4.7: Expression tree for a relational algebra expression

In Fig. 4.7 we see the above steps represented as an expression tree. The two selection nodes correspond to steps (1) and (2). The intersection node corresponds to step (3), and the projection node is step (4).

Alternatively, we could represent the same expression in a conventional, linear notation, with parentheses. The formula

$$\pi_{title,year}\Big(\sigma_{length \geq 100}(\texttt{Movie}) \cap \sigma_{studioName=\texttt{'Fox'}}(\texttt{Movie})\Big)$$

represents the same expression.

Incidentally, there is often more than one relational algebra expression that represents the same computation. For instance, the above query could also be written by replacing the intersection by logical `AND` within a single selection operation. That is,

Equivalent Expressions and Query Optimization

All database systems have a query-answering system, and many of them are based on a language that is similar in expressive power to relational algebra. Thus, the query asked by a user may have many *equivalent expressions* (expressions that produce the same answer, whenever they are given the same relations as operands), and some of these may be much more quickly evaluated. An important job of the query "optimizer" discussed briefly in Section 1.2.3 is to replace one expression of relational algebra by an equivalent expression that is more efficiently evaluated.

$$\pi_{title,year}\Big(\sigma_{length\geq 100 \text{ AND } studioName=\text{'Fox'}}(\texttt{Movie})\Big)$$

is an equivalent form of the query. □

Example 4.11: One use of the natural join operation is to recombine relations that were decomposed to put them into BCNF. Recall the decomposed relations from Example 3.32:[1]

> `Movie1` with schema {`title`, `year`, `length`, `filmType`, `studioName`}
> `Movie2` with schema {`title`, `year`, `starName`}

Let us write an expression to answer the query "Find the stars of movies that are at least 100 minutes long." This query relates the `starName` attribute of `Movie1` with the `length` attribute of `Movie2`. We can connect these attributes by joining the two relations. The natural join successfully pairs only those tuples that agree on `title` and `year`; that is, pairs of tuples that refer to the same movie. Thus, $\texttt{Movie1} \bowtie \texttt{Movie2}$ is an expression of relational algebra that produces the relation we called `Movie` in Example 3.32. That relation is the non-BNCF relation whose schema is all six attributes and that contains several tuples for the same movie when that movie has several stars.

To the join of `Movie1` and `Movie2` we must apply a selection that enforces the condition that the length of the movie is at least 100 minutes. We then project onto the desired set of attributes: `title` and `year`. The expression

$$\pi_{title,year}\Big(\sigma_{length\geq 100}(\texttt{Movie1} \bowtie \texttt{Movie2})\Big)$$

expresses the desired query in relational algebra. □

[1] Remember that the relation `Movie` of that example has a somewhat different relation schema from the relation `Movie` that we introduced in Section 3.9 and used in Examples 4.2, 4.3, and 4.4.

4.1.8 Renaming

In order to control the names of the attributes used for relations that are constructed by one or more applications of the relational algebra operations, it is often convenient to use an operator that explicitly renames operations. We shall use the operator $\rho_{S(A_1,A_2,\ldots,A_n)}(R)$ to rename a relation R. The resulting relation has exactly the same tuples as R, but the name of the relation is S. Moreover, the attributes of the result relation S are named $A_1, A_2, \ldots, A_n$, in order from the left. If we only want to change the name of the relation to S and leave the attributes as they are in R, we can just say $\rho_S(R)$.

Example 4.12: In Example 4.5 we took the product of two relations R and S from Fig. 4.3 and used the convention that when an attribute appears in both operands, it is renamed by prefixing the relation name to it. These relations R and S are repeated in Fig. 4.8.

Suppose, however, that we do not wish to call the two versions of B by names $R.B$ and $S.B$; rather we want to continue to use the name B for the attribute that comes from R, and we want to use X as the name of the attribute B coming from S. We can rename the attributes of S so the first is called X. The result of the expression $\rho_{S(X,C,D)}(S)$ is a relation named S that looks just like the relation S from Fig. 4.3, but its first column has attribute X instead of S.

When we take the product of R with this new relation, there is no conflict of names among the attributes, so no further renaming is done. That is, the result of the expression $R \times \rho_{S(X,C,D)}(S)$ is the relation $R \times S$ from Fig. 4.3, except that the five columns are labeled A, B, X, C, and D, from the left. This relation is shown in Fig. 4.8.

As an alternative, we could take the product without renaming, as we did in Example 4.5, and then rename the result. The expression $\rho_{RS(A,B,X,C,D)}(R \times S)$ yields the same relation as in Fig. 4.8, with the same set of attributes, but the relation has a name, RS, which the result relation in Fig. 4.8 does not. □

4.1.9 Dependent and Independent Operations

Some of the operations that we have described in Section 4.1 can be expressed in terms of other relational-algebra operations. For example, intersection can be expressed in terms of set difference:

$$R \cap S = R - (R - S)$$

That is, if R and S are any two relations with the same schema, the intersection of R and S can be computed by first subtracting S from R to form a relation T consisting of all those tuples in R but not S. We then subtract T from R, leaving only those tuples of R that are also in S.

The two forms of join are also expressible in terms of other operations. Theta-join can be expressed by product and selection:

A	B
1	2
3	4

Relation R

B	C	D
2	5	6
4	7	8
9	10	11

Relation S

A	B	X	C	D
1	2	2	5	6
1	2	4	7	8
1	2	9	10	11
3	4	2	5	6
3	4	4	7	8
3	4	9	10	11

Result $R \times \rho_{S(X,C,D)}(S)$

Figure 4.8: Two relations and their product

$$R \bowtie_C S = \sigma_C(R \times S)$$

The natural join of R and S can be expressed by starting with the product $R \times S$. We then apply the selection operator with a condition C of the form

$$R.A_1 = S.A_1 \text{ AND } R.A_2 = S.A_2 \text{ AND} \cdots \text{AND } R.A_n = S.A_n$$

where $A_1, A_2, \ldots, A_n$ are all the attributes appearing in the schemas of both R and S. Finally, we must project out one copy of each of the equated attributes. Let L be the list of attributes in the schema of R followed by those attributes in the schema of S that are not also in the schema of R. Then

$$R \bowtie S = \pi_L\Big(\sigma_C(R \times S)\Big)$$

Example 4.13: The natural join of the relations U and V from Fig. 4.5 can be written in terms of product, selection, and projection as:

$$\pi_{A,U.B,U.C,D}\Big(\sigma_{U.B=V.B \text{ AND } U.C=V.C}(U \times V)\Big)$$

That is, we take the product $U \times V$. Then we select for equality between each pair of attributes with the same name — B and C in this example. Finally, we project onto all the attributes except one of the B's and one of the C's; we have chosen to eliminate the attributes of V whose names also appear in the schema of U.

For another example, the theta-join of Example 4.9 can be written

$$\sigma_{A<D \text{ AND } U.B \neq V.B}(U \times V)$$

That is, we take the product of the relations U and V and then apply the condition that appeared in the theta-join. □

The redundancies mentioned in this section are the only "redundancies" among the operations that we have introduced. The six remaining operations — union, difference, selection, projection, product, and renaming — form an independent set, none of which can be written in terms of the other five.

4.1.10 Exercises for Section 4.1

Exercise 4.1.1: In this exercise we introduce one of our running examples of a relational database schema and some sample data. The database schema consists of four relations, whose schemas are:

```
Product(maker, model, type)
PC(model, speed, ram, hd, cd, price)
Laptop(model, speed, ram, hd, screen, price)
Printer(model, color, type, price)
```

The `Product` relation gives the manufacturer, model number and type (PC, laptop, or printer) of various products. We assume for convenience that model numbers are unique over all manufacturers and product types; that assumption is not realistic, and a real database would include a code for the manufacturer as part of the model number. The `PC` relation gives for each model number that is a PC the speed (of the processor, in megahertz), the amount of RAM (in megabytes), the size of the hard disk (in gigabytes), the speed of the CD reader (e.g., 4x), and the price. The `Laptop` relation is similar, except that the screen size (in inches) is recorded in place of the CD speed. The `Printer` relation records for each printer model whether the printer produces color output (true, if so), the process type (laser, ink-jet, or dry), and the price.

Some sample data for the relation `Product` is shown in Fig. 4.9. Sample data for the other three relations is shown in Fig. 4.10. Manufacturers and model numbers have been "sanitized," but the data is typical of products on sale at the end of 1996.

Write expressions of relational algebra to answer the following queries. For the data of Figs. 4.9 and 4.10, show the result of your query. However, your answer should work for arbitrary data, not just the data of these figures. *Hint*:

maker	*model*	*type*
A	1001	pc
A	1002	pc
A	1003	pc
B	1004	pc
B	1006	pc
B	3002	printer
B	3004	printer
C	1005	pc
C	1007	pc
D	1008	pc
D	1009	pc
D	1010	pc
D	2001	laptop
D	2002	laptop
D	2003	laptop
D	3001	printer
D	3003	printer
E	2004	laptop
E	2008	laptop
F	2005	laptop
G	2006	laptop
G	2007	laptop
H	3005	printer
I	3006	printer

Figure 4.9: Sample data for Product

model	*speed*	*ram*	*hd*	*cd*	*price*
1001	133	16	1.6	6x	1595
1002	120	16	1.6	6x	1399
1003	166	24	2.5	6x	1899
1004	166	32	2.5	8x	1999
1005	166	16	2.0	8x	1999
1006	200	32	3.1	8x	2099
1007	200	32	3.2	8x	2349
1008	180	32	2.0	8x	3699
1009	200	32	2.5	8x	2599
1010	160	16	1.2	8x	1495

(a) Sample data for relation PC

model	*speed*	*ram*	*hd*	*screen*	*price*
2001	100	20	1.10	9.5	1999
2002	117	12	0.75	11.3	2499
2003	117	32	1.00	10.4	3599
2004	133	16	1.10	11.2	3499
2005	133	16	1.00	11.3	2599
2006	120	8	0.81	12.1	1999
2007	150	16	1.35	12.1	4799
2008	120	16	1.10	12.1	2099

(b) Sample data for relation Laptop

model	*color*	*type*	*price*
3001	true	ink-jet	275
3002	true	ink-jet	269
3003	false	laser	829
3004	false	laser	879
3005	false	ink-jet	180
3006	true	dry	470

(c) Sample data for relation Printer

Figure 4.10: Sample data for relations of Exercise 4.1.1

For the harder expressions, it may be helpful to define one or more intermediate relations in terms of the given relations (`Product`, etc.) and then use these relations in a final expression. You can then substitute for the intermediate relations in your final expression, to get an expression in terms of the given relations.

* a) What PC models have a speed of at least 150?

b) Which manufacturers make laptops with a hard disk of at least one gigabyte?

c) Find the model number and price of all products (of any type) made by manufacturer B.

d) Find the model numbers of all color laser printers.

e) Find those manufacturers that sell Laptops, but not PC's.

*! f) Find those hard-disk sizes that occur in two or more PC's.

! g) Find those pairs of PC models that have both the same speed and RAM. A pair should be listed only once; e.g., list (i, j) but not (j, i).

*!! h) Find those manufacturers of at least two different computers (PC's or laptops) with speeds of at least 133.

!! i) Find the manufacturer(s) of the computer (PC or laptop) with the highest available speed.

!! j) Find the manufacturers of PC's with at least three different speeds.

!! k) Find the manufacturers who sell exactly three different models of PC.

Exercise 4.1.2: Draw expression trees for each of your expressions of Exercise 4.1.1.

Exercise 4.1.3: This exercise introduces another running example, concerning World War II capital ships. It involves the following relations:

```
Classes(class, type, country, numGuns, bore, displacement)
Ships(name, class, launched)
Battles(name, date)
Outcomes(ship, battle, result)
```

Ships are built in "classes" from the same design, and the class is usually named for the first ship of that class. The relation `Classes` records the name of the class, the type (`bb` for battleship or `bc` for battlecruiser), the country that built the ship, the number of main guns, the bore (diameter of the gun barrel, in inches) of the main guns, and the displacement (weight, in tons). Relation `Ships` records the name of the ship, the name of its class, and the year in which

the ship was launched. Relation `Battles` gives the name and date of battles involving these ships, and relation `Outcomes` gives the result (sunk, damaged, or ok) for each ship in each battle.

Figures 4.11 and 4.12 give some sample data for these four relations.[2] Note that, unlike the data for Exercise 4.1.1, there are some "dangling tuples" in this data, e.g., ships mentioned in `Outcomes` that are not mentioned in `Ships`.

Write expressions of relational algebra to answer the following queries. For the data of Figs. 4.11 and 4.12, show the result of your query. However, your answer should work for arbitrary data, not just the data of these figures.

a) Give the class names and countries of the classes that carried guns of at least 16-inch bore.

b) Find the ships launched prior to 1921.

c) Find the ships sunk in the battle of the North Atlantic.

d) The treaty of Washington in 1921 prohibited capital ships heavier than 35,000 tons. List the ships that violated the treaty of Washington.

e) List the name, displacement, and number of guns of the ships engaged in the battle of Guadalcanal.

f) List all the capital ships mentioned in the database. (Remember that all these ships may not appear in the `Ships` relation.)

! g) Find the classes that had only one ship as a member of that class.

! h) Find those countries that had both battleships and battlecruisers.

! i) Find those ships that "lived to fight another day"; they were damaged in one battle, but later fought in another.

Exercise 4.1.4: Draw expression trees for each of your expressions of Exercise 4.1.3.

* **Exercise 4.1.5:** What is the difference between the natural join $R \bowtie S$ and the theta-join $R \bowtie_C S$ where the condition C is that $R.A = S.A$ for each attribute A appearing in the schemas of both R and S?

! **Exercise 4.1.6:** An operator on relations is said to be *monotone* if whenever we add a tuple to one of its arguments, the result contains all the tuples that it contained before adding the tuple, plus perhaps more tuples. Which of the operators described in this section are monotone? For each, either explain why it is monotone or give an example showing it is not.

[2]Source: J. N. Westwood, *Fighting Ships of World War II*, Follett Publishing, Chicago, 1975 and R. C. Stern, *US Battleships in Action*, Squadron/Signal Publications, Carrollton, TX, 1980.

class	*type*	*country*	*numGuns*	*bore*	*displacement*
Bismarck	bb	Germany	8	15	42000
Iowa	bb	USA	9	16	46000
Kongo	bc	Japan	8	14	32000
North Carolina	bb	USA	9	16	37000
Renown	bc	Gt. Britain	6	15	32000
Revenge	bb	Gt. Britain	8	15	29000
Tennessee	bb	USA	12	14	32000
Yamato	bb	Japan	9	18	65000

(a) Sample data for relation Classes

name	*date*
North Atlantic	5/24-27/41
Guadalcanal	11/15/42
North Cape	12/26/43
Surigao Strait	10/25/44

(b) Sample data for relation Battles

ship	*battle*	*result*
Bismarck	North Atlantic	sunk
California	Surigao Strait	ok
Duke of York	North Cape	ok
Fuso	Surigao Strait	sunk
Hood	North Atlantic	sunk
King George V	North Atlantic	ok
Kirishima	Guadalcanal	sunk
Prince of Wales	North Atlantic	damaged
Rodney	North Atlantic	ok
Scharnhorst	North Cape	sunk
South Dakota	Guadalcanal	damaged
Tennessee	Surigao Strait	ok
Washington	Guadalcanal	ok
West Virginia	Surigao Strait	ok
Yamashiro	Surigao Strait	sunk

(c) Sample data for relation Outcomes

Figure 4.11: Data for Exercise 4.1.3

name	*class*	*launched*
California	Tennessee	1921
Haruna	Kongo	1915
Hiei	Kongo	1914
Iowa	Iowa	1943
Kirishima	Kongo	1915
Kongo	Kongo	1913
Missouri	Iowa	1944
Musashi	Yamato	1942
New Jersey	Iowa	1943
North Carolina	North Carolina	1941
Ramillies	Revenge	1917
Renown	Renown	1916
Repulse	Renown	1916
Resolution	Revenge	1916
Revenge	Revenge	1916
Royal Oak	Revenge	1916
Royal Sovereign	Revenge	1916
Tennessee	Tennessee	1920
Washington	North Carolina	1941
Wisconsin	Iowa	1944
Yamato	Yamato	1941

Figure 4.12: Sample data for relation Ships

! **Exercise 4.1.7:** Suppose relations R and S have n tuples and m tuples, respectively. Give the minimum and maximum numbers of tuples that the results of the following expressions can have.

* a) $R \cup S$.

b) $R \bowtie S$.

c) $\sigma_C(R) \times S$, for some condition C.

d) $\pi_L(R) - S$, for some list of attributes L.

! **Exercise 4.1.8:** The *semijoin* of relations R and S, denoted $R \ltimes S$, is the set of tuples of R that agree with at least one tuple of S on all attributes that are common to the schemas of R and S. Give three different expressions of relational algebra that are equivalent to $R \ltimes S$.

!! **Exercise 4.1.9:** Let R be a relation with schema

$$(A_1, A_2, \ldots, A_n, B_1, B_2, \ldots, B_m)$$

and let S be a relation with schema $(B_1, B_2, \ldots, B_m)$; that is, the attributes of S are a subset of the attributes of R. The *quotient* of R and S, denoted $R \div S$, is the set of tuples t over attributes $A_1, A_2, \ldots, A_n$ (i.e., the attributes of R that are not attributes of S) such that for every tuple s in S, the tuple ts, consisting of the components of t for $A_1, A_2, \ldots, A_n$ and the components of s for $B_1, B_2, \ldots, B_m$, is a member of R. Give an expression of relational algebra, using the operators we have defined previously in this section, that is equivalent to $R \div S$.

4.2 A Logic for Relations

There is another approach to expressing queries about relations that is based on logic rather than algebra. Interestingly, the two approaches (logical and algebraic) lead to the same class of queries that can be expressed. The logical query language introduced in this section is called *Datalog* ("database logic"); it consists of if-then rules. In one of these rules, we express the idea that from certain combinations of tuples in certain relations we may infer that some other tuple is in some other relation, or in the answer to a query.

4.2.1 Predicates and Atoms

Relations are represented in Datalog by symbols called *predicates*. Each predicate takes a fixed number of arguments, and a predicate followed by its arguments is called an *atom*. The syntax of atoms is just like that of function calls in conventional programming languages; for example $P(x_1, x_2, \ldots, x_n)$ is an atom consisting of the predicate P with arguments $x_1, x_2, \ldots, x_n$.